Fast Internet Packet Filtering on any Number of Attributes via Multi-dimensional Point Stabbing

Marco Pellegrini*

IMC B4-01-11

December 2001

Abstract

Internet and Intranet packet classification and filtering is governed by sets of rules, each rule specifying the range of applicability for each packet attribute (for example source address number, destination address number, source port, destination port, etc..) a priority and an action. Eppstein and Muthukrishnan [EM01] have shown that, with reasonable assumptions, the problem of finding the rule applicable to the incoming packet can be solved in the RAM model in optimal time $O(1)$ using storage $O(n^{1+\epsilon})$ for any real parameter $\epsilon > 0$ on a database of n rules. This result is restricted to rules with two attribute ranges and an extension to the multi-attribute case is left as an open problem. Here we show that the same asymptotic complexity bound holds for any fixed number d of attributes. Moreover the data structure is dynamic and has low preprocessing time. As a corollary of the techniques used, we also show that, under the same general assumptions, orthogonal range counting queries over a set of n points with bounded integer coordinates can be answered in time $O(1)$ in any fixed dimension d using storage $O(n^{1+\epsilon})$.

*Istituto di Matematica Computazionale - CNR, Pisa, Italy. pellegrini@imc.pi.cnr.it

1 Introduction

Orthogonal range searching, point stabbing and related problems, in dimension two and higher, have been an active area of research in Computational Geometry up to the early '90's leading to many efficient algorithms as well as lower bounds in several models of computation[1]. Research in this area has been recently revived by two concurrent trends. On one end several problems useful in application areas ranging from internet and intranet packet classification and filtering, object oriented run-time support and string searching have been shown to be equivalent to point stabbing queries on sets of axis oriented hyper-rectangle, with the additional twist that input coordinates are drawn from a bounded universe of integers. The second trend is the redefinition of many basic algorithms for sorting, searching and basic data structures in the unit cost RAM with bounded word size[2].

The present research is motivated by an open problem stated by Eppstein and Muthukrishnan in [EM01]. After showing that the Packet Classification Problem, which is basic in Quality of Service Internet Routing as well as for Intranet firewalls, is equivalent to retrieving a maximum priority hyper-rectangle stabbed by a query point among a data-set of hyper-rectangles, they describe an efficient algorithm for such a classification based on two attributes, but leave as an open problem the issue of achieving a similar performance in the case when $d > 2$ attributes are involved.

Here we show that, under the same assumptions used in [EM01], it is indeed possible to solve point stabbing queries, and hence the Packet Classification Problem, over an arbitrary but fixed number d of attributes in time $O(1)$ using storage $O(n^{1+\epsilon})$, for any real parameter $\epsilon > 0$, where n is the number of rules (hyper-rectangles), thus extending effectively the asymptotic bound in [EM01] to arbitrary dimension[3]. The constants in the big-Oh depend on ϵ and d.

From a technical point of view the general approach in [EM01] is to devise an efficient *dynamic* 1-dimensional data structure, make it persistent, and than apply a sweep-line along the second dimension, recording modifications, to get at the end a static 2-dimensional data structure. It is shown in [EM01] that this can be done without increasing the storage or query time with respect to the static 1D case. If one tries to extend this approach to dimension greater than two one faces the rather un-natural task of thinking about events in a multi-dimensional time. For this reason we drop this approach altogether and we resort to a static multi-dimensional divide-and-conquer line of attack, which is amenable to the use of induction arguments over the dimensionality of the problems.

The second result shown in this paper concerns the problem of preprocessing a set of n data points in U^d, where $U = [0..2^w - 1]$, into a data structure so that for a query hyper-rectangle the number of points in the hyper-rectangle is computed efficiently. Using our data structure for point-stabbing as a stepping-stone we can answer queries in time $O(1)$ using storage $O(n^{1+\epsilon})$. Using the data structure in [FMdB99] as basis one can achieve query

[1] See [Aga97] and [AE98] for up to date surveys.

[2] A different trend involves examining those basic problems in external memory models.

[3] To be precise both the main technical result in [EM01] and in this paper hold for input hyper-rectangles and query points with coordinates drawn in the bounded integer universe $U = [0..2^w - 1]$, and use a free parameter that allows trading off storage against query time. For sake of comparison we consider the case when the additional condition $|U| = n^{O(1)}$ holds, and in both cases we set the free parameter to a value so that the best query time is attained.

time $O(\log\log n)$ within $O(n^{1+\epsilon})$ storage. Previous results were known for data points with unrestricted coordinates. A result of Chazelle [Cha88] based on range trees achieves query time $O((\log n)^{d-1})$ using storage $O(n(\log n)^{d-2})$ but does not allow to trade-off the query time against storage. A result of Bentley and Maurer [BM80] achieves $O(\log n)$ query time using storage $O(n^{1+\epsilon})$.

In the next Section we give some background and discuss related results. The formal definition and main technical theorem is in Section 3. Some storage improvements are discussed in Section 4 and Counting Range Reporting in Section 5. Finally in Section 6 we discuss preprocessing time and dynamization.

2 Background and related results

As described in [EM01], [FM00], and references therein, a packet filter f is a d-dimensional range $[l_f^1, r_f^1] \times ... \times [l_f^d, r_f^d]$ associated to a priority p_f and an action A_f. Any packet P can be seen as a d-dimensional vector $[P_1, .., P_d]$ summarizing the relevant header information. Rule f applies to P if, for each $j = [1..d]$, $P_j \in [l_f^j, r_f^j]$. A data base F of filters is preprocessed so that any incoming packets P is classified as quickly as possible by determining the highest priority filter that applies to P[4]. The specific nature of the action will not concern us in this paper. Existing IP routers consider only the destination address in routing decisions, this corresponds to a packet filtering problem in dimension $d = 1$. However, as Quality of Service is becoming a serious concern, new proposed standards try to make much more use of packet header information to attain the overall network performance goals as well as single-user performance guarantees. The packet filtering problem should be seen as a tool onto which a variety of Service Policies are to be built upon. Similarly, in Intranet firewall application the goal is to to enforce secure access policies. Because of high peak rates of usage, the filtering time per single packet is important and for this reason we stress the optimality of the query time $O(1)$, as well as the freedom in the choice of attributes, at the expense of memory storage, although trade-off between time and storage are feasible and will be useful for fine tuning. Because of the importance of the application we take advantage of all the features of the problem. The coordinates involved are integers in a bounded universe $U = [0..2^w - 1]$. Currently for IP addresses $[0..2^{32} - 1]$. Moreover, as in [EM01], we add the technical condition that the rule set size n is not too small with respect to U, that is $n \geq |U|^{1/C}$. This is reasonable since, for example, setting $C = 4$ with $|U| = 2^{32}$ we have that the rule set should not be smaller than $2^8 = 256$, which does not represent a serious limitation[5].

Several authors have addressed the packet classification problem for $d = 2$ attributes [WSV01],[Sri01], [GM99], [LS98], [SSV99], [SVSW98], however these schemes do not scale well to higher dimensions. In general, either the query time is retained but memory consumption becomes prohibitive, as high as $O(n^d)$, or the memory consumption is linear but the query time is also close to linear in the worst case. A scheme in [FM00] called FIS

[4]In this paper we do not address the filter conflict detection problem, which consist in detecting whether it is possible or not to invoke two filters with the same priority and conflicting actions. A somewhat different notion of conflict is used in [WSV01].

[5]If this condition is dropped we can still achieve time $O(1)$, however the storage bound will depend on n and $|U|$: the (small) factor n^ϵ in the storage bound becomes $|U|^\epsilon$.

tree, based on results in [FMdB99], scales well to higher dimension and achieves best query time $O(\log\log|U|)$ using storage $O(n^{1+\epsilon})$. Gupta and McKeown [GM00] attain query time $O((\log|U|)^d \log n)$ and update time $O((\log|U|)^{d-1}\log n)$ using storage $O(n(\log|U|)^d)$, no space-time trade offs are considered. Finally as mentioned above, in [EM01] $O(1)$ query time using $O(n^{1+\epsilon})$ storage is attained, but only for $d = 2$.

In [SSV99] and [WSV01] a lower bound $\Omega((\log|U|)^{d-1})$ to the query time for d-dimensional filter lookup is proved. However this lower bound holds in a binary decision tree model[6] which is suitable for binary searching schemes but does not capture the richness of index arithmetic operations allowed in the RAM model. Indeed, to escape this lower bound, we shall use in full the features of RAM arithmetic on addresses.

Since the packet classification problem reduces to a point stabbing problem which is well known in computational geometry we recall also results from this area. The one dimensional version of the problem can be solved using predecessor queries on the data structures of Willard [Wil83] [Wil84], or the stratified trees of van Emde Boas et al. [van77], [vEBKZ77] and [Ove88]. Recent progress in devising ways of exploiting the power of unit-cost RAM models has lead to improved searching methods for inputs from a bounded universe [And96], [BF99], [ABR00b] and [BCKM01] which are now being used also in higher dimensional setting for range searching and nearest neighbour queries [ABR00a], [AEIS99], and [Wil00]. For the point stabbing problem in dimension two and higher with general (non bounded) real coordinates, Chazelle and Guibas in [CG86], as an application of fractional cascading, use storage $O(n(\log n)^{d-1})$ to achieve query time $O((\log n)^{d-1})$.

3 Point Stabbing Problems

3.1 The problem

A Hyper-rectangle in R^d is the Cartesian product of d real intervals, one on each coordinate line. If we consider a discrete set $U = [0..2^w - 1] \subset R$ we can still define in a similar way hyper-rectangles in the U^d space often referred to as a multi-dimensional grid. The problem we discuss is the following. Given an input set H of n hyper-rectangles with coordinates in U, build a data structure $D(H)$ so that for a query point $q \in U^d$ we can retrieve efficiently properties of the stabbed set $H_q = \{h \in H | q \in h\}$. Standard properties studied in literature are:

(a) A listing of the set H_q (reporting query).

(b) The cardinality of H_q (counting query).

(c) If we associate, via a function f, to each element in H an element of a commutative semigroup (S,+), then compute $\sum_{h \in H_q} f(h)$. Usually it is assumed that each semigroup-sum operation requires time $O(1)$. (semigroup sum query).

[6] In the binary decision tree model the algorithm is modelled as a binary decision tree, where the leaves represent different configurations that must be distinguished for the algorithm to be correct, and the length of the longest path from the root to any leaf is the worst case time lower bound.

(d) If we associate via a function g to each element in H an element of a totally ordered set (S,$\geq$), then compute $\max_{h \in H_q} g(h)$. Usually it is assumed that each comparison operation requires time $O(1)$. (max query).

3.2 Main Result

Theorem 1 *For an integer w, let $U = [0,..,2^w - 1]$ be the set of binary numbers of w bits. Let H be a set of n hyper-rectangles in U^d and k a parameter, $1 \leq k \leq n$. We can build a data structure $D(H)$ using storage $O(nk^{f(d)}(\log_k |U|)^d)$ answering queries (b), (c) and (d) in time $O((\log_k |U|)^d)$ and query (a) in time $O((\log_k |U|)^d + |H_q|)$, where the constants hidden in the big-Oh notation depend on d.*

Remark: The function $f(d)$ grows as $d^2/2$.
Proof. For simplicity of exposition we describe the data structure focussing on the max operation. Let us consider the **case** $d = 1$.
We build recursively a k-ary tree where each node x is associated to an interval $I(x)$ in U. For convenience k will be a power of two. The root is associated to U. Recursively starting from the root we partition $I(x)$ into k intervals of length $|I(x)|/k$ and we associate each interval to each child of x. An hyper-rectangle $h \in H$ covers x if $I(x) \subset h$, crosses x if end-point of h is in $I(x)$. Denote with $H(x)$ the input at x, with $CR(x)$ the set of all elements of $H(parent(x))$ crossing x, with $CO(x)$ all elements in $H(parent(x))$ covering x. Call $g(x)$ the max element over $CO(x)$ and store $g(x)$ at x.
At the root $H(root) = H$, $CO(root) = \emptyset$, $CR(root) = H$. Recursively for $x \neq root$ we set $H(x) = CR(x)$.
Storage. We have at most $\log_k |U|$ levels in the tree. At each level the endpoints of the intervals in H are partitioned among the sets $H(x)$ for all x in the same level, so at each level there are at most $2n$ nodes with non empty input (we call these nodes *active*). So in total there are $O(n \log_k |U|)$ nodes. Each node uses storage $O(k)$. In total the tree uses storage $O(nk \log_k |U|)$.
Query time. Starting from the root, after visiting node x we visit the child y of x such that $q \in I(y)$. We compute on the fly the max $g(x)$ value over all visited nodes. Since we visit a node per level we visit $O(\log_k |U|)$ nodes. At each node in time $O(1)$ we can select the next node just by computing the index $(q - start(I(x)))$ *div* k. Pointers to children are stored in a direct access array.

This is essentially the starting static 1-dimensional result in [EM01] which we restate in a different language so to make clearer the line of reasoning leading to the multi-dimensional extensions . Let us now consider the **case** $d = 2$.
We build recursively a k^2-ary tree where each node x is associated to a square $I(x)$ in U^2. For convenience k will be a power of two. The root is associated to U^2. Recursively starting from the root we partition $I(x)$ into k^2 squares of area $|I(x)|/k^2$ and we associate each square to each child of x. An hyper-rectangle $h \in H$ covers x if $I(x) \subset h$, crosses x if an end-point of h is in $I(x)$, horizontally (resp. vertically) crosses x if nether cover nor crosses but an horizontal (resp. vertical) edge of h meets the boundary of $I(x)$. Note that the 4 classes we have defined are mutually exclusive.
Denote with $H(x)$ the input at x, with $CR(x)$ the set of all elements of $H(parent(x))$ crossing x, with $CO(x)$ all elements in $H(parent(x))$ covering x, with $HC(x)$ the elements in

$H(parent(x))$ horizontally covering x, with $VC(x)$ the elements in $H(parent(x))$ vertically covering x. At the root $H(root) = H$, $CO(root) = \emptyset$, $CR(root) = H$. Recursively for $x \neq root$ we set $H(x) = CR(x)$.
Call $g(x)$ the max element over $CO(x)$ and store $g(x)$ at x. For each node x we build the 1-dimensional structure for the sets $HC(x)$ and $VC(x)$ as described above, calling them $D_1(HC(x))$ and $D_1(VC(x))$.
Query time. Starting from the root, after visiting node x we visit the child y of x such that $q \in I(y)$. We compute on the fly the max $g(x)$ value over all visited nodes and the results of the queries with the x-projection of q in $D_1(VC(x))$ and with the y-projection of q in $D_1(HC(x))$. The number of nodes visited is proportional to the height of the tree $O(\log_{k^2} |U|^2) = O(\log_k |U|)$ and at each node we make two $O(\log_k U)$ time queries to auxiliary data structures. Finding and accessing the child of a node containing the query point q is done in $O(1)$ time with easy arithmetic. So the total query time is $O(\log_k^2 |U|)$.
Storage. The height of the tree is $O(\log_k |U|)$, however since at each level the n vertices of the hyper-rectangles in H are partitioned among the nodes, there are at most $4n$ active nodes at each level. Each node has $O(k^2)$ pointers for children and as many pointers to auxiliary data structures. So the main tree, without auxiliary data structures, has size $O(nk^2 \log_k |U|)$.
Consider now all active nodes at level $l-1$, and let us estimate the the total size of the sets $HC(y)$ at level l. A single input hyper-rectangle h at $x = parent(y)$ contributes to at most $2k$ sets $HC(.)$ among all nodes that are children of x. Moreover, at level $l-1$, h appears only 4 times since vertices are partitioned. Denoting with m_i the cardinality of $HC(.)$ of node i at level l we have that for every i $m_i < n$, moreover summing on level l $\sum_i m_i \leq 8kn$. All auxiliary data structures at level l cost order of:

$$\sum_i m_i k \log_k |U| \leq 8k^2 n \log_k |U|.$$

Summing over all levels we have that the total size of all auxiliary data structures is: $O(nk^2(\log_k |U|)^2)$. So the overall storage is $O(nk^2(\log_k |U|)^2)$.
General Case. The argument is inductive on the dimension. We assume we can build data structures for dimensions from 1 up to $d-1$ which inductively use storage $O(mk^{f(d-1)}(\log_k |U|)^{d-1})$ to answer queries in time $O(\log_k^{d-1} |U|)$ and we show how to build the data structure in dimension d.
We build recursively a k^d-ary tree where each node x is associated to a hyper-square $I(x)$ in U^d. For convenience k will be a power of two. The root is associated to U^d. Recursively starting from the root we partition $I(x)$ into k^d squares of measure $|I(x)|/k^d$ and we associate each hyper-square to each child of x. An hyper-rectangle $h \in H$ covers x if $I(x) \subset h$, crosses x if a corner vertex of h is in $I(x)$. The following lemma is central.

Lemma 1 *Answering a d-dimensional query q for the collection H of hyperboxes that neither cover nor cross, but intersect the hyper-cube Q is equivalent to answering d $(d-1)$-queries over data sets obtained by partitioning H into d groups and applying an orthogonal projection of each group onto a Cartesian $(d-1)$-plane.*

Proof. For simplicity of exposition we assume general position. Consider the hyperbox Q and the interval I_i^Q that is the projection of Q into the i-th coordinate axis. We have that

Q is equivalent to the Cartesian product of all its projections onto d Cartesian axis:

$$Q = I_1^Q \times ... \times I_d^Q.$$

Moreover let q be a point in R^d and q_i its projection onto the i-th axis. We have that $q \in Q$ if and only if $q_i \in I_i^Q$ for all $i = 1, .., d$. Similarly we define the hyperbox H as the Cartesian product of its projection onto the coordinate axis.
We have the following property: $q \in Q \cap H$ if and only of for all $i = 1, .., d$: $q_i \in I_i^Q \cap I_i^H$. Moreover $Q \subset H$ iff for all $i = 1, .., d$: $I_i^Q \subset I_i^H$. This corresponds to the covering condition. Also $Q \cap H \neq \emptyset$ iff for all $i = 1, .., d$: $I_i^Q \cap I_i^H \neq \emptyset$.
Suppose that $\exists i | I_i^Q \subset I_i^H$, then since every vertex of H will project to one of the endpoints of I_i^H and both are external to I_i^Q then no vertex of H is in Q.
Suppose now that $\forall i | I_i^Q \not\subset I_i^H$, since also we assume that $\forall i I_i^Q \cap I_i^H \neq \emptyset$, we have that in each projection at least an end-point of I_i^H is in I_i^Q. Since the Cartesain product of any pick of an end-point from all I_i^H gives a vertex of H we have that this vertex is in Q.
Therefore the above condition is both necessary and sufficient. So if we are in the situation when no vertex of H is in Q then there is always an i so that $I_i^Q \subset I_i^H$. for such fixed i to decide if a $q \in Q$ is also in H we need to test only $d-1$ projections, which corresponds to projecting both q and H orthogonally onto the Cartesian hyperplane orthogonal to the direction i. So the problem is reduced to a point location in an hyper-box one dimension lower.
If we have a collection of hyper-boxes H which meet Q but neither cover it nor cross it then we can partition such a collection into d sub-collection according to the value of the index i for which the above condition holds, breaking ties arbitrarily. To answer d-dimensional queries in each collection is equivalent to answering a $d-1$-dimensional query in each sub-collection. ∎
Now we resume the proof of the main theorem.
Query time. Starting from the root, after visiting node x we visit the child of x such that $q \in I(x)$. We compute on the fly the max $g(x)$ value over all visited nodes and the results of the queries with the projection of q on the Cartesian flats and their associated data structures. The number of nodes visited is proportional to the height of the tree $O(\log_{k^d} |U|^d) = O(\log_k |U|)$ and at each node we make at most d queries to auxiliary data structures. Finding and accessing the child of a node containing the query point q is done in $O(1)$ time with easy arithmetic. So the total query time is

$$O(\log_k |U|) \times d \times O((\log_k |U|)^{d-1}) = O((\log_k |U|)^d)$$

Storage. The height of the tree is $O(\log_k |U|)$, however since at each level the $2^d n$ vertices of the hyper-rectangles in H are partitioned among the nodes, there are at most $2^d n$ active nodes at each level. Each node has $O(k^d)$ pointers for children and as many pointers to auxiliary data structures. So the main tree, without auxiliary data structures, has size $O(nk^d \log_k |U|)$.
Consider now all active nodes at level $l-1$, and let us estimate the the total size of the input sets at level l. A single input hyper-rectangle h at $x = parent(y)$ contributes to at most $2dk^{d-1}$ sets among all nodes that are children of x. Moreover, at level $l-1$, h

appears only 2^d times since vertices are partitioned. Denoting with m_i the cardinality of the input sets of node i at level l we have that, for every i, $m_i < n$. Summing on level l: $\sum_i m_i \leq 2^{d+1} d k^{d-1} n$. All auxiliary data structures at level l cost order of:

$$\sum_i m_i k^{f(d-1)} (\log_k |U|)^{d-1} \leq 2^{d+1} d k^{d-1} n k^{f(d-1)} (\log_k |U|)^{d-1}.$$

Now defining f recursively as $f(1) = 1$ and $f(d) = f(d-1) + d - 1$ we have a bound: $O(nk^{f(d)} (\log_k |U|)^{d-1})$. Summing over all levels we have that the total size of all auxiliary data structures is: $O(nk^{f(d)} (\log_k |U|)^d)$.

Reporting queries. In the case of max, semigroup sum and counting we need storing only $O(1)$ non-recursive information for each non-active child to answer the query. Reporting queries need a bit more work. We store the full list of covering hyper-boxes. However at each node, such list is not longer than the the input of his parent. Since as observed before the total input at each level is bounded by $2^d n$ The total storage cost per level is at most $O(k^d 2^d n)$ and there are $O(\log_k |U|)$ levels. Note that this extra storage is asymptotically of the same order of the size of the tree itself without auxiliary data structures. ∎

Now, choosing $k = n^\epsilon$ and using the additional assumption $n > |U|^{1/C}$ we have the following corollary:

Corollary 1 *For an integer w, let $U = [0, .., 2^w - 1]$ be the set of binary numbers of w bits. Let H be a set of n hyper-rectangles in U^d, and $n \geq |U|^{1/C}$. We can build a data structure $D(H)$ using storage $O(n^{1+\epsilon})$ answering queries (b), (c) and (d) in time $O(1)$ and query (a) in time $O(1 + |H_q|)$, where the constants hidden in the big-Oh notation depend on d, ϵ and C.*

4 Improvements for $d = 1$

By applying a trick in [Wil83] we reduce the storage and query time for the case $d = 1$. We take the input set E of $2n$ end-points, we sort them and we select one point every l points, so that the number of endpoints is reduced to $2n/l$, Let S be this reduced set. Then we split every input segment containing more than two points of S into at most 3 sub-segment, one maximal sub-segment with endpoints in S, and two sub-segments with at most one endpoint in S and at least one in $E \setminus S$. Now we build the parametric tree of Theorem (1) on the endpoints in S. For each node that is empty non-active we have at most l original points in the interval of that node, and therefore at most l original segments or pieces of segment. We build for such a set a linear size data structure with $O(\log l)$ query time. Note that the points in $E \setminus S$ are partitioned among the non-active leaves so the total storage of these data structures is $O(n)$. The total storage is:

$$O((n/l) k \log_k |U|) + O(n).$$

The query time is:

$$O(\log_k |U| + \log l).$$

Choosing $l = k \log_k |U|$. We get storage $O(n)$ and query time :

$$\log_k |U| + \log k + \log\log_k |U|.$$

This is minimized when $\log k = \log |U| / \log k$, so $\log k = \sqrt{\log |U|}$. Thus we have a method with storage $O(n)$ and query time $O(\sqrt{\log |U|})$. The same asymptotic complexity for the 1D case is attained in [Wil84] however we feel that our treatment is somewhat simpler.

5 Counting range queries

An interesting consequence of the technical Theorem (1) is that we can answer counting range searching queries much faster than what was previously reported in literature. In the range counting problem for $U = [0..2^w - 1]$ the input data are points in U^d and the queries are hyper-rectangles with coordinates in U. It is well known that general such query in R^d can be reduced to 2^d dominance queries [EO82]. In a dominance query we count the points in an octant of pre-defined sign sequence so that the apex of the octant is a sufficient specification given the fact that the sign sequence is uniform for all queries. Now, in dominance queries we can reverse the role of data and queries: that is we can store octant (of opposite sign sequence) and query with just the apex, instead of storing points and querying with a the full octant. So we can reduce the problem to point stabbing counting queries for a special type of hyper-rectangle data and answer efficiently apex queries using the data structure of Theorem (1). When the universe is small compared to the input set, i.e. $n > |U|^{1/C}$, we can apply directly the Corollary (1) and solve the counting query in time $O(1)$ using $O(n^{1+\epsilon})$ storage.

For large universe we first reduce the problem to rank space, so that after the reduction we have a universe $U' = [1..n]$ which satisfies the above condition. Calling $T(n, U)$ the time to answer rank queries in a set of n elements form a universe U, we can answer counting queries in time $O(dT(n, U) + 1)$ and storage $O(n^{1+\epsilon})$. For the reduction to rank space we can use any of the recent fast methods for 1-dimensional predecessor queries [And96], [BF99], [BCKM01].

6 Preprocessing and Dynamization

We show how to build in time $O(n^{1+\epsilon})$ the data structure for the reporting queries in the setting of Corollary (1). This bound holds for all four types of queries. We start from an bare tree skeleton and insert recursively each input hyper-rectangle in turn, starting from the root. For an hyper-rectangle h and a child y of the root, we spend time $O(1)$ to decide whether h is classified intersecting, covering, crossing or in which sub-class of Lemma 1 should be put. We spend time $O(1)$ in updating the max field $g(y)$ if it is needed. So the time spent is not more than the worst case storage used locally at the children of the root. This implies that the time analysis follows the same steps of the worst case storage analysis, therefore the same global bound holds for time too.

It is possible to add or delete a filter from the data set in time $O(n^{\epsilon})$, while retaining the constant query time, as long as the branching factor k^d is not changed, and $n > |U|^{1/C}$ holds. This is done by storing the local lists used for answering report queries as priority queues with respect to the "priority" field of the filters. The max priority query on each list is answered in $O(1)$ time per node visited, so still in $O(1)$ time altogether without an increase of storage. The preprocessing is increased by a $O(\log n)$ factor.

Our method is very simple and self contained, relying only on uniform grids and on integer division in the query phase to quickly determine the grid cells containing a query point, thus we avoid sophisticated hashing strategies and the even more sophisticated dynamic hashing routines required in the dynamic setting. We plan as future work to implement the data structure and test its practical effectiveness.

References

[ABR00a] S. Alstrup, G. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *FOCS 2000*, pages 198–207, 2000.

[ABR00b] S. Alstrup, G. S. Brodal, and T. Rauhe. Optimal static range reporting in one dimension. Technical Report 3, The IT University of Copenhagen, November 2000. To appear in STOC 2001.

[AE98] P.K. Agarwal and J. Erickson. Geometric range searching and its relatives. In Advances in Discrete and Computational Geometry (B. Chazelle, E. Goodman, and R. Pollack eds.), American Mathematical Society, Providence, 1998.

[AEIS99] A. Amir, A. Efrat, P. Indyk, and H. Samet. Efficient regular data structures and algorithms for location and proximity problems. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.

[Aga97] P. K. Agarwal. Range searching. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.

[And96] Arne Andersson. Faster deterministic sorting and searching in linear space. In *37th Annual Symposium on Foundations of Computer Science*, pages 135–141, Burlington, Vermont, 14–16 October 1996. IEEE.

[BCKM01] A. Brodnik, S. Carlsson, J. Karlsson, and J. I. Munro. , worst case constant time priority queue. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms*, pages 523–528, New York, NY, USA, January 2001. ACM Press.

[BF99] P. Beame and F. E. Fich. Optimal bounds for the predecessor problem. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1999.

[BM80] J. L. Bentley and H. A. Maurer. Efficient worst-case data structures for range searching. *Acta*, 13:155–168, 1980.

[CG86] Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1(2):163–191, 1986.

[Cha88] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, June 1988.

[EM01] D. Eppstein and S. Muthukrishnan. Internet packet fileter management and rectangle geometry. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms*, pages 827–835, New York, NY, USA, January 2001. ACM Press.

[EO82] Herbert Edelsbrunner and Mark H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters*, 14(3):124–127, May 1982.

[FM00] A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *Proceedings of INFOCOM, volume 3*, pages 1193–1202. IEEE, March 2000.

[FMdB99] P. Ferragina, S. Muthukrishnan, and M. de Berg. Multi-method dispatching: a geometric approach with applications to string matching. In *Proceedigns of STOC '99*, 1999.

[GM99] P. Gupta and N. McKeown. Packet classification on multiple fields. In *SIGCOMM*, pages 147–160, 1999.

[GM00] P. Gupta and Nick McKeown. Dynamic algorithms with worst-case performance for packet classification. In *Proc. IFIP Networking 2000, Paris, France*, 2000.

[LS98] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *SIGCOMM*, pages 203–214, 1998.

[Ove88] Mark H. Overmars. Efficient data structures for range searching on a grid. *Journal of Algorithms*, 9(2):254–275, June 1988.

[Sri01] V. Srinivasan. A packet classification and filtre management system. In *Prceedings of IEEE INFOCOM*, 2001. To appear.

[SSV99] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. In *SIGCOMM*, pages 135–146, 1999.

[SVSW98] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *SIGCOMM*, pages 191–202, 1998.

[van77] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, April 1977.

[vEBKZ77] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.

[Wil83] Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, August 1983.

[Wil84] Dan E. Willard. New trie data structures which support very fast search operations. *Journal of Computer and System Sciences*, 28(3):379–394, June 1984.

[Wil00] Dan E. Willard. Examining computational geometry, Van Emde Boas trees, and hashing from the perspective of the fusion tree. *SIAM Journal on Computing*, 29(3):1030–1049, June 2000.

[WSV01] P. Warkhede, S. Suri, and G. Varghese. Fast packet classification for two-dimensional conflict -free filters. In *Prceedings of IEEE INFOCOM*, 2001. To appear.